Parallel Tabu Search Algorithm for the Permutation Flow Shop Problem with Criterion of Minimizing Sum of Job Completion Times

W. Bożejko, and J. Pempera
Institute of Computer Engineering, Control and Robotics
Wroclaw University of Technology, Wroclaw, Poland
{wojciech.bozejko, jaroslaw.pempera}@pwr.wroc.pl

***Abstract* —This paper deals with an intelligent algorithm dedicated for the use in manufacturing systems. Particularly, it develops the fast parallel tabu search algorithm to minimize sum of job completion times in the flow shop scheduling problem. So called multimoves are used, that consist in performing several independent moves simultaneously, which allow one to guide very quickly the search process to promising areas of the solutions space, where good solutions can be found. Besides, an adaptable dynamic tabu list and varying neighborhood are proposed to avoid being trapped at a local optimum. The proposed algorithms are experimentally evaluated on a personal computer with duo-core processor and found to be relatively more effective in finding solutions of quality better than other leading approaches, and also it makes in a much shorter time. The presented ideas can be extended to cover search methods for other hard problems.**

***Keywords* — intelligent manufacturing, flow-shop, parallel computing, tabu search, experimental evaluation.**

I. Introduction

THE flow shop scheduling problems models naturally many real-life manufacturing systems, since there are many practical as well as important applications for a job to be processed in series with more one-stage in industry [1–3]. This paper considers the flow shop scheduling problem to minimize sum of job completion times (mean completion or flow time) described as follows. A specified number of jobs are to be processed on a specified number of machines. Each job must go through all the machines in exactly the same order and the job order is the same on every machine. Each machine can process at most one job at any point in time, and each job may be processed on at most one machine at any time. The objective is to find a schedule that minimizes the mentioned above criterion. The problem is indicated in literature by $F||C_{sum}$ and it is strongly NP-hard.

II. Problem description and preliminaries

The flow-shop problem can be formulated as follows. There are a set of n jobs $J=\{1,2,\ldots,n\}$ and a set of machines $M=\{1,2,\ldots,m\}$. Each of n jobs from the set J has to be processed on m machines $1,2,\ldots,m$ in that order.

Thus job j, $j\in J$ consists of a sequence of m operations; each of them corresponding to the processing of job j on machine k during an uninterrupted processing time $p_{jk}>0$. Machine k, $k\in M$ can execute at most one job at a time, each job can be processed on at most one machine, and it is assumed that each machine processes the jobs in the same order. A feasible schedule is defined by completion times C_{jk}, $j\in J$, $k\in M$ of job j on machine k, such that the above constraints are satisfied. For the given processing order represented by permutation $\pi=(\pi(1),\ldots,\pi(n))$ on set J, the feasible schedule (small as possible) can be found by using the following recursive formulae:

$$C_{\pi(j),k} = \max(C_{\pi(j-1),k}, C_{\pi(j),k-1}) + p_{\pi(j),k}, \qquad (1)$$

calculated for $j \in J, k \in M$, where $\pi(0)=0$, $C_{j,0}=0$ $j\in J$, $C_{0,k}$ $k\in M$. Let Π denote the set of all permutations defined on the set J. We wish to find such permutation $\pi^*\in\Pi$, that

$$C_{sum}(\pi^*) = \min_{\pi\in\Pi} C_{sum}(\pi), \qquad (2)$$

where $C_{sum}(\pi) = \sum_{j=1}^{n} C_{\pi(j),m}$ is the sum of the job completion times.

There are plenty of good heuristic algorithms for solving flow shop problem with the objective of minimizing maximal job's completion times (C_{max}). For the sake of special properties(blocks of critical path, [4]) it is recognized as an easier one than a problem with objective C_{sum}. Unfortunately, there are not any similar properties (which can speedup computations) for the C_{sum} flow shop problem. Constructive algorithms (LIT and SPD from [5], NSPD [6]) have low efficiency and can only be applied to a limited range. There is hybrid algorithm in [7], consisting of elements of tabu search, simulated annealing and path relinking methods. The results of this algorithm, applied to Taillard benchmark tests [8], are the best known ones in the literature nowadays. The big disadvantage of

the algorithm is its time-consumption. Parallel computing is the way to speed it up. This problem was introduced in [9] on the example of parallel simulated annealing.

III. PARALLEL TABU SEARCH METHOD (TS)

Currently, tabu search approach, (see Glover [10] and [11]), is one of the most effective methods using local search techniques to find near-optimal solutions of many scheduling problems. This technique aims to guide the search by exploring the solution space of a problem beyond local optimality.

The main idea of this method involves starting from an initial basic job permutation and searching through its neighborhood, a set of permutations generated by the moves, for a permutation with the lowest value of optimality criterion. The search then is repeated starting from the best permutation, as a new basic permutation, and the process is continued. One of the main ideas of tabu search algorithm is the use of a tabu list to avoid cycling, overcoming local optimum, or continuing the search in a too narrow region and to guide the search process to the solutions regions which have not been examined. The tabu list records some attributes of the performed moves and/or basic solution. The elements of this list, for the current iteration, determine the subset of forbidden solution. A move having prohibited attributes is forbidden, although, it can be performed if it is sufficiently profitable.

The list content is refreshed each time a new basic permutation is found; the oldest element is removed and the new one is added.

There are two basic types of tabu search parallelization discussed in literature. The first one, called single-walk, is based on neighborhood decomposition onto concurrent working processors. Received solutions are exactly the same as in sequential algorithm, but computing time is shorter. Aarts and Verhoeven [12] make the distinction between single-step and multiple-step parallelism within this type. In the case of single-step implementations, neighbors are searched and evaluated in parallel after neighborhood partitioning. The algorithm subsequently selects and performs one single move. In multiple-step parallelizations, a sequence of consecutive moves in the neighborhood is made simultaneously.

The second type of parallelization, called multiple-walk type, is based on concurrent working tabu search threads, running on different processors. There are dwo sub-types of this parallelizm: independent search, where there is no communication between threads, and cooperative search, with exchanging e.g. the best known solution found of the thread. Classification of multiple-walk tabu search algorithm was created by Voss in [13]. The first parallel implementations of tabu search based on this type of strategy seem to concern the quadratic assignment problem and job shop scheduling Taillard [14].

We propose a hybrid type of tabu search parallelism in this paper.). Proposed new parallel asynchronous tabu search uses a list of current solutions instead of one current solution in classical tabu. Such a parallel algorithm is based on two-level parallelism. One level is based on concurrently explored solutions from the list by parallel working processors. These solutions are explored concurrently to find the best representative (solution) of each block, and added to list, and this is the second level of parallelism, made by another group of parallel working processors. Of course we also use a tabu list, to prevent generating solutions on main list serially. The second level of parallelism does not have any influence on results of computations (only speed up), but the first level of parallelism does.

In a proposed algorithm, the several diversification components is applied that assists us additionally to avoid getting trapped at a local optimum. These components have been proposed by Grabowski, Pempera and Wodecki [15−17], where they were successfully applied on those very fast tabu search algorithms for the flow shop and job shop problems.

The algorithm tabu search terminates when a given number of iterations has been reached without improvement of the best current goal function value, the algorithm has performed a given number of iterations (*Maxiter*), time has run out, etc.

A. Moves and neighborhood

We implement basically two well-known types of the neighborhoods for the permutation scheduling problems, namely $N^{INS}(\pi)$ and $N^{ICH}(\pi)$. The first neigbourhood is based on insertion moves *INS*. The insert move operates on a sequence of jobs on a machine and removes a job placed at a position in this sequence and inserts it in another position of the sequence. More precisely, let $v=(a,b)$ be a pair of jobs, $a,b\in\{1, 2,...,n\}$, $a\neq b$. The pair $v=(a,b)$ defines a move in π. This move consists in removing job a from its original position $ps(a)$, and next inserting it in the position immediately after job b (or before b) in π if $ps(a)<ps(b)$ (or $ps(a)>ps(b)$).

The second neigbourhood is based on interchange moves *ICH*. The interchange move can be described by the pair $v=(a,b)$ be a pair of jobs, $a,b\in\{1, 2,...,n\}$, $a\neq b$. The pair $v=(a,b)$ defines a move in π. This move consists in removing job a and b from they original positions, $ps(a)$ and $ps(b)$, and next inserting it in the positions $ps(b)$ and $ps(a)$ respectively.

The move v generates the new permutation π_v. The all possible moves both types generates neighborhoods of cardinality $(n\text{-}1)^2$ and $(n\text{-}1)^2/2$ respectively.

B. Multimoves

A multimove consists of several moves that are performed simultaneously in a single iteration of algorithm. The performances of the multimoves allow us to generate permutations that differ in various significant ways from those obtained by performing a single move and to carry the search process to hitherto non-visited regions of the solution space. In local search algorithms, the use of multimoves can be viewed as a way to apply a

mixture of intensification and diversification strategies in the search process.

For the set of all feasible moves V defined for the permutation π and corresponding with them neighborhood $N(V,\pi)$, let $B(\pi)\subseteq V$ be a set of moves, which generates solution better than π, i.e. .

$$B(\pi)=\{v\in V : C_{sum}(\pi_v)<C_{sum}(\pi)\} \quad (3)$$

Based on the set $B(\pi)$, we create a subset of independent moves $I(\pi)$, called multimove $\bar{v}$. Roughly speaking, two moves are called independent, if they are separated to each other by least one job in π. The greedy method of constricting the multimove, initially sorts the elementary moves in non-decreasing order of the solution quality and next, beginning at the first, each moves it examined to be independent with all previous moves. If it is not independent, it is deleted from the set $B(\pi)$, see in [17] for details. Based on the definition of independent moves, we define k-separated moves. Two moves are k-separated, if they are separated to each other by least k job in π.

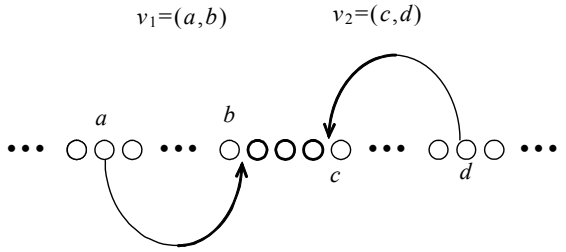

Fig. 1 Two moves 3-separated.

The intuition following from the definition of $\bar{v}$ suggests that $\pi\bar{v}$ should be significantly better than π_v generated by the best (single) move $v\in\bar{v}$, since the total improvement of $C_{\text{sum}}(\pi\bar{v})$ can be obtained by combining all the improvements produced by the individual moves from $\bar{v}$. It allows the algorithm to achieve very good solutions in a much shorter time. Therefore, the performance of multimove $\bar{v}$ guides the search to visit new more promising regions of the solution space where good solutions can found.

C. Search process

Similarly to in other local search algorithms, our TS starts from an initial basic permutation π that implies neighborhood $N(V,\pi)$. This neighborhood is searched in the in the following manner.

First, the best move $v^*\in V$ that generates the permutation $\pi_{v^*}\in N(V,\pi)$ with the lowest C_{sum} is chosen, i.e.

$$C_{sum}(\pi_{v^*}) = \min_{v\in V} C_{sum}(\pi_v), \quad (4)$$

If $C_{sum}(\pi_{v^*})<C_{sum}(\pi_{BF})$ (where π_{BF} is the best solution found so far), then the move v^* is selected for the search process. Otherwise, i.e. if $C_{\text{sum}}(\pi_{v^*})\geq C_{\text{sum}}(\pi_{BF})$, then the set of unforbidden moves (UF) that do not have the tabu status, is selected. Finally, from the set unforbriden moves the best is chosen for the search. If all moves are forbidden (a very rare case), then the oldest element of tabu list is deleted and the search is repeated until a unforbidden move is found.

Similarly to in Grabowski and Pempera [17], when at least *Piter* consecutive non-improving iterations pass in the algorithm, the multimove is created and performed. Algorithm starts with *INS* neighbourhoor, and each time then the multimove is executed switch to *ICH* and inversely.

D. Tabu list and tabu status of move

In our algorithms we use two type of neigboorhoods, so we need some universal attributes come from both type of moves. The tabu mechanism is implemented as a cyclic list T with length *LengthT* containing ordered pairs of jobs. The list T is a realization of theshort-term search memory.

If the insert move $v=(a,b)$ is performed on permutation π, then the pair of jobs $(a,\pi(ps(a)+1))$, if $ps(a)< ps(b)$, or the pair $(\pi(ps(a)-1),a)$ otherwise is added to T, the function $ps(a)$ returns the position of job a in permutation π. If the interchange move $v=(a,b)$ is performed on permutation π, then the pair of jobs (a,b). Each time before adding a new element to T, we must remove the oldest one.

With respect to a permutation π, let us $x=ps(a)$ and $y=ps(b)$, a move $(a,b)\in INS$ is forbidden, i.e. it has tabu status, if $A(a)\cap\{\pi(x+1),\pi(x+2),...,\pi(y)\}\neq\emptyset$, if $x<y$, and $B(a)\cap\{\pi(y),\pi(y+1),...,\pi(x-1)\}\neq\emptyset$, otherwise, where

$$A(j)=\{i\in J : (j,i)\in \mathrm{T}\}, \quad (3)$$

$$B(j)=\{i\in J : (j,i)\in \mathrm{T}\}. \quad (4)$$

With respect to a permutation π, let us $x=ps(a)$ and $y=ps(b)$, a move $(a,b)\in ICH$ is forbidden, i.e. it has tabu status, if $A(a)\cap\{\pi(x+1),\pi(x+2),...,\pi(y)\}\neq\emptyset$, and $B(a)\cap\{\pi(y),\pi(y+1),...,\pi(x-1)\}\neq\emptyset$

Set $A(j)$ (or set $B(j)$) indicates which jobs are to be processed after (or before) job j with respect to the current content of the tabu list T.

As mentioned above, our algorithms use a tabu list with dynamic length. This length is changed, as the current iteration number *iter* increases. The length change is used as a "pick" intended to carry the search to another area of the solutions space. It can be viewed as a specific disturbance that gives an additional assistance to avoid getting trapped at a local optimum of algorithm.

In this tabu list, length *LengthT* is a cyclic function defined by the expression where l=1,2,... is the number of the cycle, $W(l)=\sum_{s=1}^{l} H(s-1)+(l-1)\times h$ (here H(0)=0), and *LTS* is the value taken from [15] equal to $6+[n/(10m)]$, where $[x]$ represents the integer of x. Further, h is the width of the pick equal to $2\times LTS$, and $H(l)$ is the interval between the neighbour picks equal to 6×LTS. If *LengthT* decreases then a suitable number of the oldest elements of

tabu list T is deleted and the search process is continued.

If a multimove $\bar{v}$ is performed, then the attributes of the move $v^* \in \bar{v}$ with the smallest value of $C_{sum}(\pi_{v^*})$ is added to tabu list T.

IV. Computational results

In this section we report the results of empirical tests to evaluate the relative effectiveness of the proposed tabu search algorithms. In this paper, we tested our several version of the proposed algorithms TS: one thread, multirun, and parallel.

One thread version of the algorithm was indeed a sequential algorithm which was implemented to compare other versions. In a multirun version the same algorithm was executed a number of times and the best solution of all exactions was fixed as a result – so it can be understand as a independent parallel algorithm (without communication). Real parallelization was used in a parallel version of the algorithm – running threads exchanges the information of the best solution found.

The algorithms TS were coded in Personal 6.0 Builder C++, run on a PC with Intel Core 2 Duo 2.66 GHz processor and the WINDOWS XP operating system, and tested on the first 5 groups of benchmark instances provided by Taillard [18]. The benchmark set contains 120 particularly hard instances of 12 different sizes, selected from a large number of randomly generated problems. For each size (group) $n \times m$: 20×5, 20×10, 20×20, 50×5, 50×10, a sample of 10 instances was provided.

Each run of TS algorithm need an initial permutation, which can found by any method. In our tests, we use algorithm NEH [19] to generate initial solution for the pivotal the run, the remaining initial solution (multirun, parallel), we generate by performing $n/4$ random interchange moves.

In our tests, TS are terminated after performing a number *Maxiter* =10 000 of iterations on each instance, additionally for the one thread version we test TS on the 20 000 iteration. The parallel and multirun TS version, for the each instances performs two threads simultaneously on the two cores of processors. Execution of the program are controlled by the operating system.

The value of tuning parameter *Piter* for TS is drawn from [15] equal to 3. All multimoves are 2-separated. The effectiveness of the algorithms was analyzed in both terms of CPU time and solution quality.

For each test instance, we collected the following values

- $PRD(\pi^A) = 100\% \left(\dfrac{C_{sum}(\pi^A) - C_{sum}(\pi^{ref})}{C_{sum}(\pi^{ref})} \right)$

  – the value of the percentage relative difference between C_{sum} function value for the reference solution taken from [7] and solution produced by the algorithm A.
- BCT – computation time, where the best solution are found (in seconds).
- TCT – total computation time (in seconds).

For the each group instances the mean values of the above values are calculate and shown in Table 1.

TABLE 1: COMPUTATIONAL RESULTS

Group	*PARALLEL*			*MULTIRUN*		
	PRD	*BCT*	*TCT*	*PRD*	*BCT*	*TCT*
20×5	0.000	0.2	3.4	0.007	0.2	3.4
20×10	0.000	0.6	7.0	0.004	0.6	6.9
20×20	0.000	2.3	13.6	0.000	0.9	13.3
50×5	0.372	34.1	50.6	0.339	30.9	50.0
50×10	0.600	75.7	105.5	0.497	67.2	104
Average	0.194			0.169		

As we can see in Table 1 and 2 the best results (in average) were obtained by multirun version of the algorithm (for all 50 instances), but parallel version was better than multirun for first 30 instances (0.0% of PRD). Times of computation (and cost of computation, as a sum of iterations executed on each processor) was comparable. Table 2. Shows that sequential (using one thread) version of the algorithm was significantly worse than parallel and multirun. Even increasing of the number of iterations from 10000 to 20000 did not notably improved the average PRD.

TABLE 1: (CONTINUATION).

Group	*NEH*	*ONE THREAD*					
		10000			**20000**		
	PRD	*PRD*	*TCT*		*PRD*	*BCT*	*TCT*
20×5	5.235	0.007	3.1		0.007	0.3	6.3
20×10	4.593	0.000	6.6		0.000	2.0	13.0
20×20	4.183	0.010	12.6		0.010	1.6	25.1
50×5	8.691	1.003	46.6		0.870	52.0	93.3
50×10	7.404	1.378	99.0		1.217	122	198
Average	6.021	0.479			0.421		

V. Conclusion

In order to decrease the computational effort for the search, we propose to use the multimoves that consist in performing several moves simultaneously in a single iteration of algorithms and guide the search process to more promising areas of the solutions space, where "good solutions" can be found. It allows the algorithm to achieve very good solutions in a much shorter time. Also, we propose a tabu list with dynamic length which is changed cyclically, as the current iteration number of algorithms increases, using the ``pick`` in order to avoid being trapped at a local optimum.

Computational experiments are given and compared with the results yielded by the best algorithms discussed in the literature. These results show that the proposed algorithm provides better results than attained by the leading approaches. Nevertheless, some improvements in our algorithm are possible. For instance, attempts to calculate the lower bounds on the goal function instead of computing it explicitly for selecting the best solution and to refine the multimoves may induce a further improvement of the computational results. It might be

interesting to develop new more sophisticated neighborhoods, and to combine them in the series and/or parallel structures, creating new algorithms.

The results obtained encourage us to extend the ideas proposed to the problems with different objective functions or to other sequencing problems.

REFERENCES

[1] J. Grabowski, J. Pempera, "Sequencing of jobs in some production systems. *European Journal of Operational Research*, vol. 126, 2000, pp. 131-151.

[2] M.T.M. Rodrigues, L. Gimeno, C.A.S Passos, T. Campos, "Reactive scheduling approach for multipurpose chemical batch plants, *Computers and Chemical Engineering*, vol. 20, 1996, pp. 1215-1220.

[3] Y.D. Kim, H.G. Lim, M.W. Park, "Search heuristics for a flowshop scheduling problem in a printed board assembly process", *European Journal of Operational Research*, vol. 91, 1996 pp. 124-143.

[4] J. Grabowski, J. Pempera, "New block properties for the permutation flow-shop problem with application in TS". *Journal of Operational Research Society* vol. 52, 2001, pp. 210-220.

[5] C. Wang, C. Chu, J. Proth, "Heuristic approaches for n/m/F/$\sum C_i$ scheduling problems". *European Journal of Operational Research*, 1997, pp. 636-644.

[6] J. Liu, "A new heuristic algorithm for Csum flowshop scheduling problems", Personal Communication, 1997.

[7] R C. R.eeves, T. Yamada, "Solving the Csum Permutation Flowshop Scheduling Problem by Genetic Local Search", *IEEE International Conference on Evolutionary Computation*, 1998, pp. 230-234.

[8] E. Taillard, "Benchmarks for basic scheduling problems", *European Journal of Operational Research* vol. 64, 1993, 278-285.

[9] W. Bożejko, M. Wodecki, "The new concepts in parallel simulated annealing method", *Lecture Notes in Computer Science* No. 3070, Springer, 2004, pp. 853-859.

[10] F. Glover, "Tabu search. Part I". *ORSA Journal of Computing* 1, 1989, 190-206.

[11] F. Glover, "Tabu search. Part II". *ORSA Journal of Computing* 2, 1990, 4-32.

[12] E.H.L Aarts., M. Verhoeven, *Local search*, in Annotated bibliographies in Combinatorial Optimization (Dell'Amico M., Maffioli F., Martello S., eds.), Wiley and Sons, Chichester, 1997.

[13] S. Voss, *Tabu search: Applications and prospects*, in Network Optimization Problems (Du D.-Z., Pardalos P.M., eds.), World Scientific, 1993.

[14] E. Taillard, "Parallel taboo search techniques for the job shop scheduling problem problem", *ORSA Journal on Computing*, vol. 6, 1994, pp. 108-117.

[15] J. Grabowski, M. Wodecki, "A very fast tabu search algorithm for the flow shop problem with makespan criterion". *Computers and Operations Research* vol. 11, 2004, pp. 1891-1909.

[16] J. Grabowski, M. Wodecki, "A very fast tabu search algorithm for the job shop problem", in: C. Rego, B. Alidaee (Eds), Metaheuristic Optimization via Memory and Evolution; Tabu Search and Scatter Search, Kluwer Academic Publishers, 2005, pp. 115-144.

[17] J. Grabowski, J. Pempera, "Some local search algorithms for no-wait flow-shop problem with makespan criterion". *Computers and Operations Research*, vol. 32, 2005, pp. 2197-2212.

[18] E. Taillard, "Benchmarks for basic scheduling problems". *European Journal of Operational Research* vol. 64, 1993, pp. 278-285.

[19] M. Nawaz, E. Enscore, I. Ham, "A Heuristic algorithm for the *m*-machine, *n*-job flowshop sequencing problem". *OMEGA The International Journal of Management Science* vol. 11, 1983, pp. 91-95.